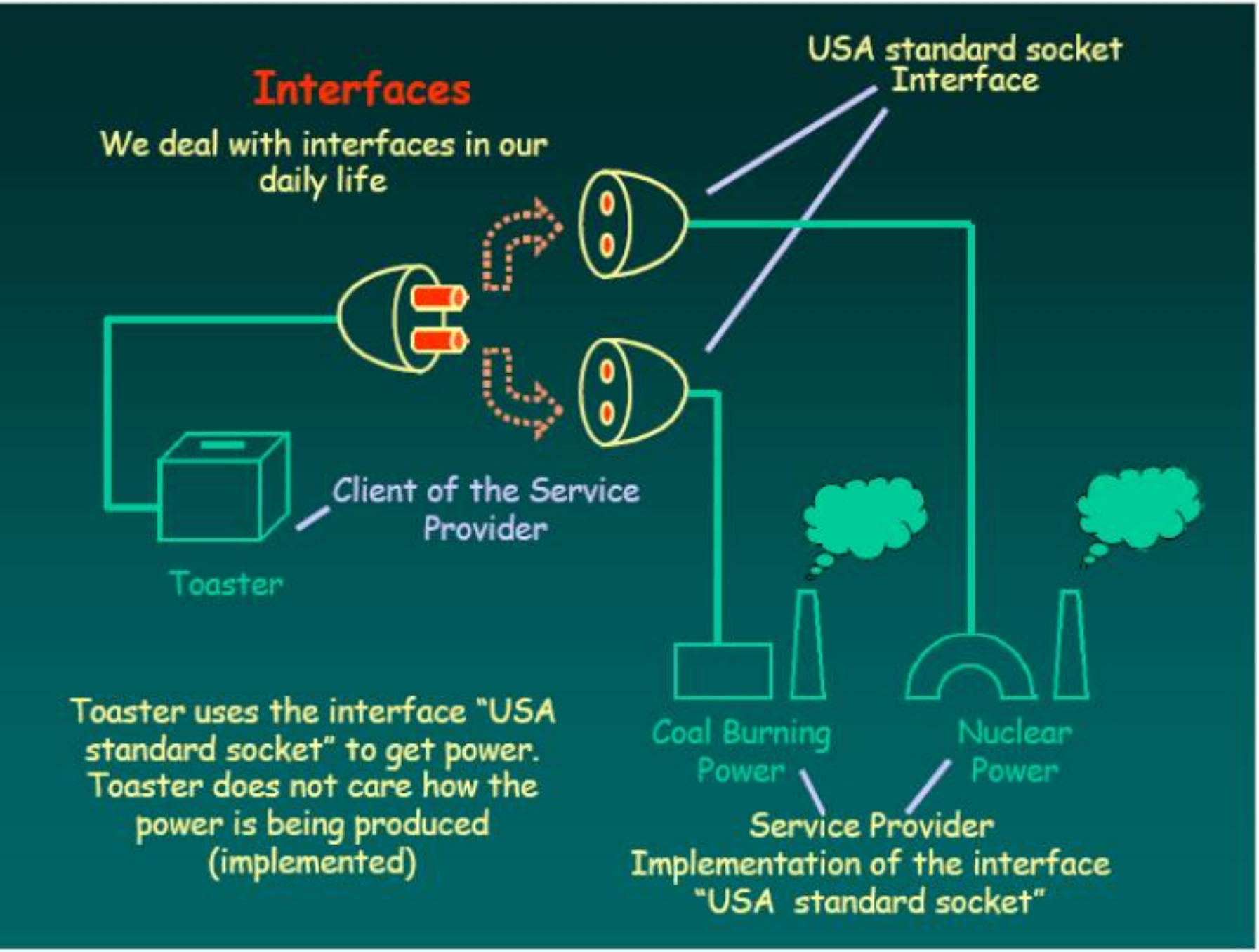
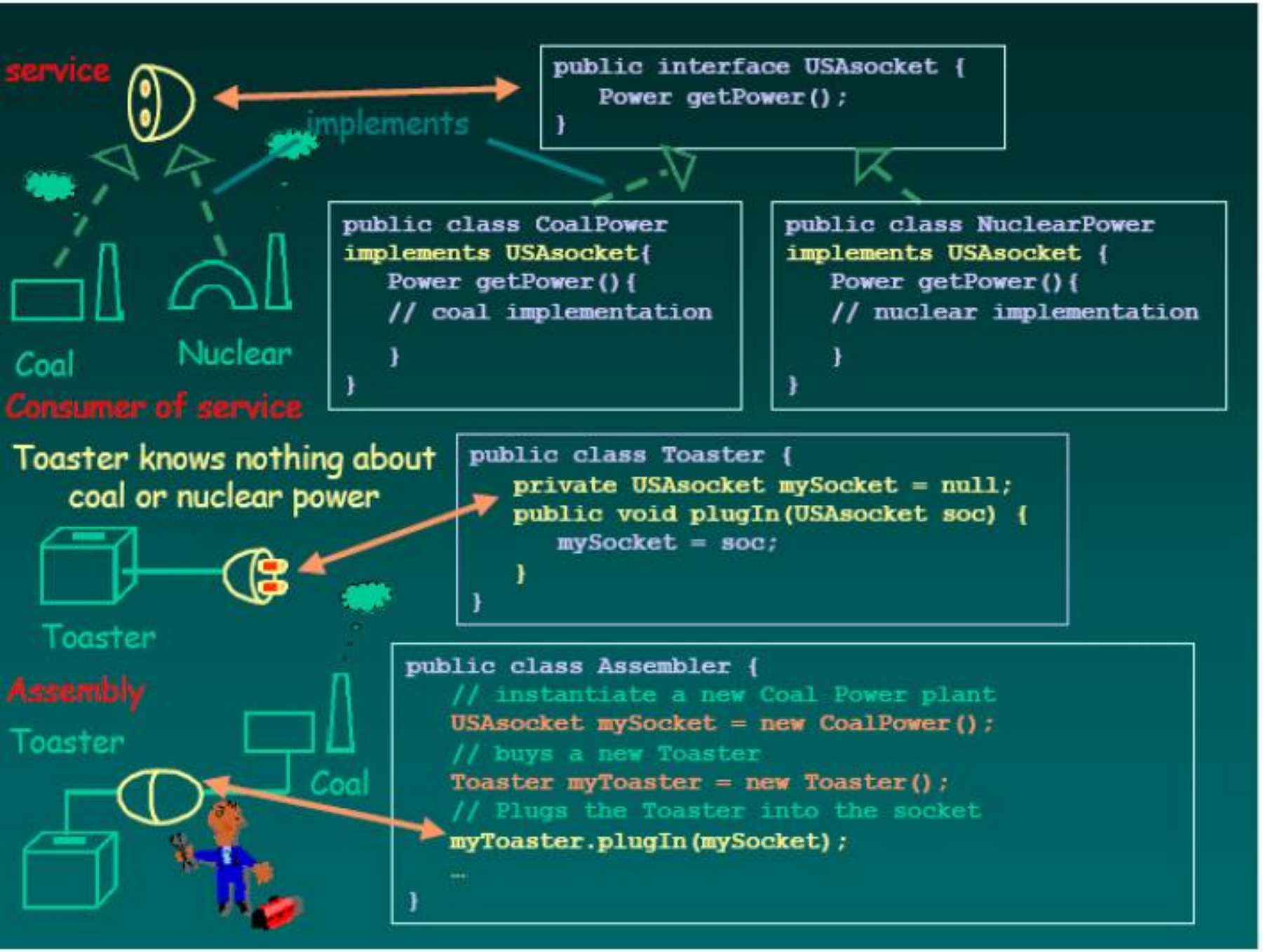
Java: Interface

IF2210 – Semester II 2020/2021

by: Yohanes Nugroho; rev: AI, SA, YW, SAR, YR





Multiple Inheritance

- Terkadang program memerlukan *multiple inheritance* (sebuah class merupakan sub class dari banyak superclass (> 1))
- Class dalam Java hanya memiliki satu superclass.
- *Multiple inheritance* dilakukan melalui penggunaan interface
- Misalkan kita ingin mencatat barang-barang berharga seseorang untuk menghitung nilai total aset
  - Melibatkan subclass `House` dari project lain (inheritance hierarchy modelling **buildings**)
  - Melibatkan subclass `Car` dari project lain (inheritance hierarchy of **vehicles**)

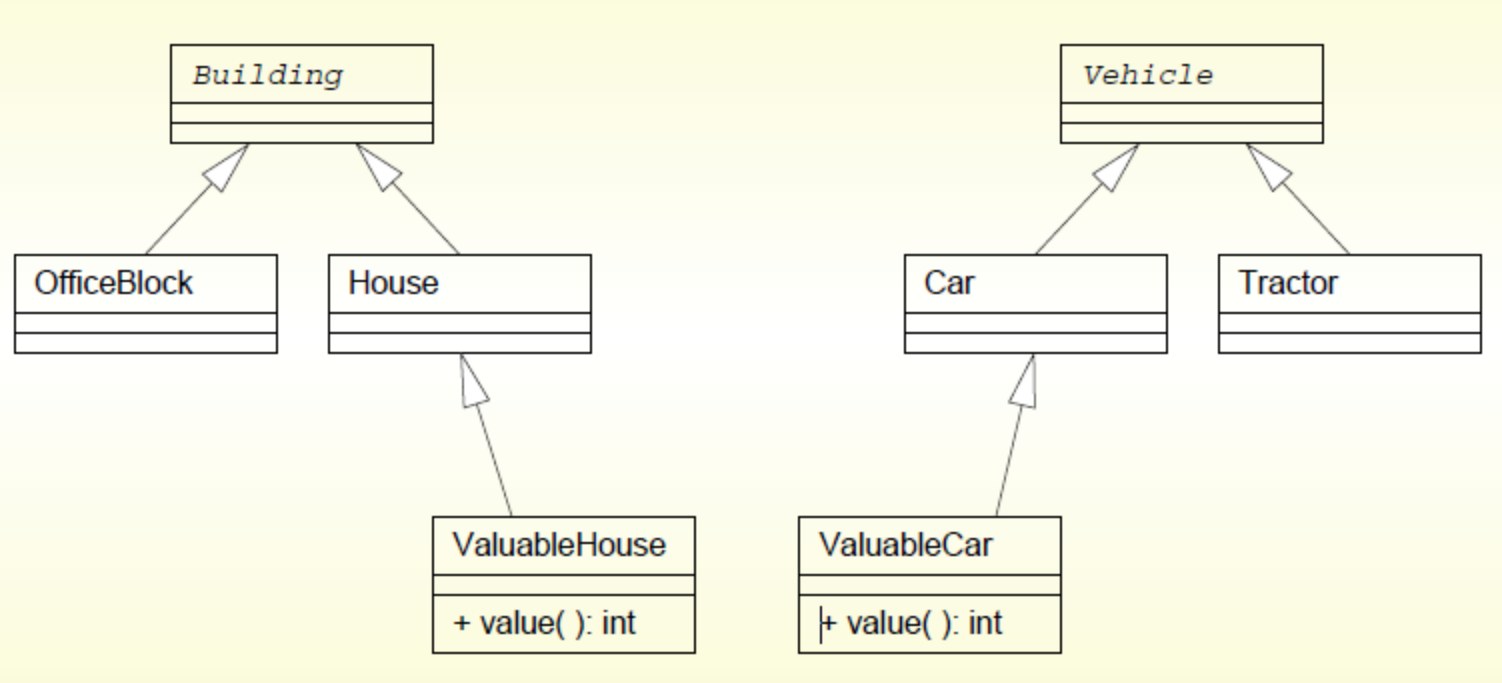
Inheritance hierarchy modelling buildings

```
public class House extends Building {
  // The number of bedrooms in the house.
  private int noOfBedrooms;

  public House(int requiredNoOfBedrooms) {
    noOfBedrooms = requiredNoOfBedrooms;
  } // House

  // Return the number of bedrooms in the house.
  public int getNoOfBedrooms() {
    return noOfBedrooms;
  } // getNoOfBedrooms
      ...
} // class House

public abstract class Building {
  ...
} // class Building

public class OfficeBlock extends Building {
  ...
} // class OfficeBlock
```

Inheritance hierarchy of vehicles

```
public class Car extends Vehicle {
  // The number of doors on the car.
  private final int noOfDoors;

  // Construct a car with a given number of doors.
  public Car(int requiredNoOfDoors) {
    noOfDoors = requiredNoOfDoors;
  } // Car

  // Return the number of doors on the car.
  public int getNoOfDoors() {
    return noOfDoors;
  } // getNoOfDoors
  ..
} // class Car

public abstract class Vehicle {
  ...
} // class Vehicle

public class Tractor extends Vehicle {
  ...
} // class Tractor
```

Tambah subclass?



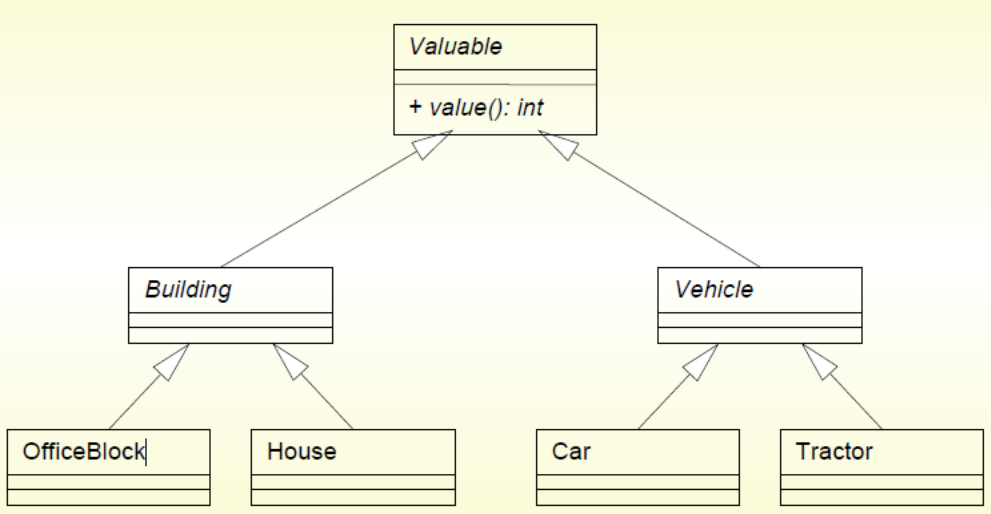
Problem, karena

- Akan perlu class lainnya dengan valuable juga, seperti: `ValuableBoat`, `ValuableArtWork`, `ValuableJewellery`. Dll.
- Bisa menghitung nilai house dan car, tapi tidak ada relasi antara keduanya
- Untuk menghitung total nilai dari beberapa valuables
  - menggunakan array of objects yang mewakili valuable item
  - masing-masing dengan `value()` instance method.
- Tipe array tersebut harus berupa `Object[]`
  - `Object` is only link between `ValuableHouse` and `ValuableCar`.
  - Not every instance of `Object` has `value()` instance method!
- Sehingga source codenya dapat menjadi seperti ini:

ValuablesFragment.java

```
Object[] valuables;
// Code here to create and populate this array.
      ....
int total = 0;
for (Object someValuable : valuables)
  if (someValuable instanceof ValuableHouse)
    total += ((ValuableHouse)someValuable).value();
  else if (someValuable instanceof ValuableCar)
    total += ((ValuableCar)someValuable).value();
  else if (someValuable instanceof ValuableArtWork)
    total += ((ValuableArtWork)someValuable).value();
  else if
    ...
```

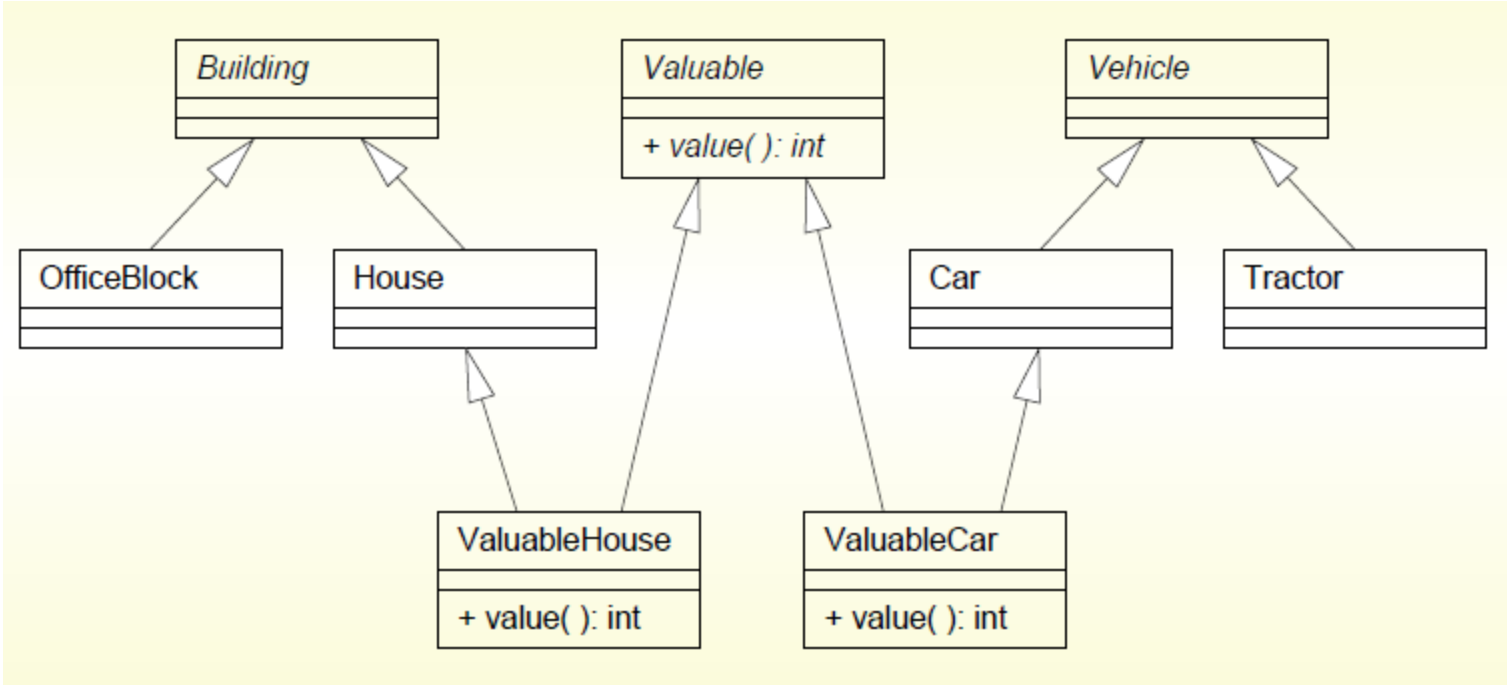
The Valuable class?



Problem?

- Tidak perlu tambahan kelas `ValuableHouse` and `ValuableCar`, tapi harus mengubah kelas yang ada
- Harus mendefinisikan juga value untuk kelas lainnya (yang sebenarnya tidak kita perlukan)
- Ide utamanya adalah: bagaimana membuat realsi antara class `ValuableHouse` dan `ValuableCar`.

The Valuable class



Problem?

- Jika dua atau lebih superclass berisi instance method dengan nama dan parameter yang sama

```
public class Super1 {
  ...
  public void methodA() {
    ...
  } // methodA
  ...
} // class Super1

public class Super2 {
  ...
  public void methodA() {
    ...
  } // methodA
  ...
} // class Super2
```

Subclass dari kedua superclass

```
public class Sub extends Super1, Super2 {
  ...
  public void methodB() {
    ...
    methodA();
    ...
  } // methodB
  ...
} // class Sub
```

Inheritance: multiple inheritance

- Ambiguity problem
  - `methodA()` mana yang dipanggil oleh `methodB()`?
- Run time efficiency problem
- Saat virtual machine melakukan dynamic method binding
  - perlu menelusuri inheritance hierarchy untuk setiap superclass
  - butuh waktu lebih daripada penelusuran single inheritance hierarchy.
- Sehingga Java tidak membolehkan class punya > 1 superclass
  - setiap class, kecuali `java.lang.Object`, punya tepat satu superclass
  - `Object` has none.

The `Valuable` interface

- Java membolehkan *partial multiple inheritance*
- Class `Valuable` berisi satu instance method, `value()`
- Cara menghitung nilai house berbeda dengan cara menghitung nilai car
- Sehingga `value()` akan menjadi abstract method, dan
- `Valuable` menjadi kelas abstrak

Interface: definition

- Sebuah interface seperti halnya class, kecuali dalam hal
  - semua instance methods harus berupa abstract methods
  - hanya method interfaces yang dideklarasikan.
- Implementasi method harus disediakan oleh setiap non-abstract class yang mengimplementasikan interface.

Interface: definition

- Instance dari StopClock adalah polymorphic
  - it is a StopClock, is a JFrame dan is an ActionListener.
- Interfaces tidak bisa berisi constructor methods

```
import java.awt.event.ActionListener;
import javax.swing.JFrame;
public class StopClock extends JFrame implements ActionListener {
  ...
  public void actionPerformed(ActionEvent event) {
    ...
  } // actionPerformed
  ...
} // class StopClock
```

The `Valuable` interface

```
// Objects which have a value obtained via a value() method.
public interface Valuable {
  // The value of this Valuable.
  int value();
} // interface Valuable
```

The ValuableHouse class

```
public class ValuableHouse extends House implements Valuable {
  // A measure of the value of the area the house is in.
  private double locationDesirabilityIndex;

  // Construct a ValuableHouse with a given number of bedrooms
  // and location desirability.
  public ValuableHouse(int requiredNoOfBedrooms, double
      requiredLocationDesirabilityIndex) {
    super(requiredNoOfBedrooms);
    locationDesirabilityIndex = requiredLocationDesirabilityIndex;
  } // ValuableHouse

  // Calculate and return the value of this valuable item.
   @Override
  public int value() {
    return (int) (getNoOfBedrooms() * 50000 * locationDesirabilityIndex);
  } // value

  // Return a short description of this as a valuable item.
  @Override
  public String toString() {
    return "House worth " + value();
   } // toString
 } // class ValuableHouse
```

The ValuableCar class

```
// Representation of a Valuable which is a car.
public class ValuableCar extends Car implements Valuable {
  // A measure of the value of the car in general.
  private double streetCredibilityIndex;

  // Construct a ValuableCar with a given number of doors
  // and general desirability.
  public ValuableCar(int requiredNoOfDoors,double requiredStreetCredibilityIndex) {
    super(requiredNoOfDoors);
    streetCredibilityIndex = requiredStreetCredibilityIndex;
  } // ValuableCar

  // Calculate and return the value of this valuable item.
  @Override
  public int value() {
    return (int) (getNoOfDoors() * 2000 * streetCredibilityIndex);
  } // value

  // Return a short description of this as a valuable item.
  @Override
  public String toString() {
    return "Car worth " + value();
  } // toString
} // class ValuableCar
```

Collection of Valuables

```
// Representation of a collection of Valuables.
public class Valuables {
  // The Valuables, stored in a partially filled array, together with size.
  private final Valuable[] valuableArray;
  private int noOfValuables;

  // Create a collection with the given maximum size.
  public Valuables(int maxNoOfValuables) {
    valuableArray = new Valuable[maxNoOfValuables];
    noOfValuables = 0;
  } // Valuables

  // Add a given Valuable to the collection (ignore if full).
  public void addValuable(Valuable valuable) {
    if (noOfValuables < valuableArray.length) {
      valuableArray[noOfValuables] = valuable;
      noOfValuables++;
    } // if
  } // addValuable

  ...
```

...lanjutan...

```
...

// Calculate and return the total value of the collection.
public int totalValue() {
  int result = 0;
  for (int index = 0; index < noOfValuables; index++)
    result += valuableArray[index].value();
  return result;
} // totalValue

// Return a short description of the collection.
@Override
public String toString() {
  if (noOfValuables == 0)
    return "Nothing valuable";
  String result = valuableArray[0].toString();
  for (int index = 1; index < noOfValuables; index++)
    result += String.format("%n%s", valuableArray[index]);
  return result;
} // toString

...
```

...lanjutan

```
  ...
  // Create a Valuables collection, add Valuable items and show result.
  // Purely for testing during development.
  public static void main(String[] args) {
    Valuables valuables = new Valuables(5);
    // My first house -- I was so proud of its spare bedroom
    // and 'value for money' area.
    valuables.addValuable(new ValuableHouse(2, 0.5));

    // My first car, not quite a 'head turner',
    // but its third door was handy when the main 2 got stuck.
    valuables.addValuable(new ValuableCar(3, 0.25));

    // It was nice to have a new car when I started work.
    valuables.addValuable(new ValuableCar(4, 1.0));

    // Then I won the lottery! (Yeah, right.)
    valuables.addValuable(new ValuableHouse(6, 2.0));
    valuables.addValuable(new ValuableCar(12, 4.0));
    System.out.println("My valuables are worth " + valuables.totalValue());
    System.out.println(valuables);
  } // main

} // class Valuables
```

Interface

- Java memiliki konsep *interface* yang memungkinkan sebagian fitur *multiple inheritance* diimplementasikan.
- *Interface* "mirip" dengan kelas abstrak yang semua *method*-nya juga abstrak.
- *Interface* tidak punya konstruktor, destruktor (*finalizer*), dan apapun, hanya punya member variabel statik atau konstan dan deklarasi *method*.
- Dengan kata lain, ***interface* tidak memiliki deskripsi *state* sebuah objek.**
  - Hanya mendeklarasikan *behavior* yang dimiliki objek.

Contoh interface

```
interface Draw {
  void draw();
  void draw3D();
}
```

Implementasi interface

- Isi interface diimplementasikan oleh kelas dengan *keyword* `implements`.
- Sebuah kelas boleh mengimplementasikan banyak *interface*.
- Contoh:

```
class Cicrle implements Draw {
  void draw() { /*implementasi draw*/ }
  void draw3D() { /*implementasi draw3D*/ }
}
```

Interface: a class can implement many interfaces

- A class can extend at most one other class
  - but may implement any number of interfaces
  - interfaces listed, with commas between,
- after reserved word **implements.**
  - E.g. `StopClock` which automatically stops and starts when mouse moved out of / back in to window…

StopClock

```
import java.awt.ActionListener;
import java.awt.MouseListener;
import javax.swing.JFrame;
      ...
public class StopClock extends JFrame
    implements ActionListener, MouseListener {

  ...

  // actionPerformed is specified in the interface ActionListener
  public void actionPerformed(ActionEvent event) {
    ...
  } // actionPerformed

  ... Various methods here, as specified in MouseListener.

} // class StopClock
```

Implementasi banyak interface

```
interface Color {
  void setColor(int color);
  int getColor();
}

class Lingkaran implements Draw, Color {
  void draw() { /*implementasi draw*/ }
  void draw3D() { /*implementasi draw3D*/ }
  void setColor(int color) { /*implementasi setColor*/ }
  int getColor() { /*implementasi getColor*/ }
}
```

Beda kelas abstrak dengan interface

- Kelas abstrak boleh memiliki *method* yang sudah diimplementasikan.
  - *Interface* harus "kosong" (tidak ada *method* yang terdefinisi pada *interface*).
- Kelas hanya boleh meng-*extend* (diturunkan dari) satu kelas.
  - Kelas boleh mengimplementasikan banyak *interface*.
- Suatu kelas boleh sekaligus `extends` dan `implements`, contoh: `class FileInputStream extends InputStream implements Closeable, AutoCloseable { ... }`

Kapan memakai kelas abstrak dan interface (1)

- Kelas abstrak:
  - Jika sudah ada algoritma yang bisa diimplementasikan di kelas tersebut.
- *Interface*:
  - Jika hanya memberi kontrak, misalnya *interface* `Measureable` untuk menyatakan objek yang bisa diukur keliling dan luasnya.
  - Jika ingin menggunakan konsep *multiple inheritance*.

Kapan memakai kelas abstrak dan interface (2)

- Kelas abstrak biasanya diturunkan menjadi kelas-kelas yang masih sangat berhubungan satu sama lain, misalnya `AbstractList` diturunkan menjadi `LinkedList`, `ArrayList`, `CopyOnWriteArrayList`, dst.
- Interface dapat diimplementasi oleh kelas-kelas yang tidak berhubungan, misalnya `Comparable` yang diimplementasikan oleh berbagai kelas, mulai dari `Integer`, `String`, hingga `Date/Time`.

Observer Pattern